Workload-Aware Views Materialization for Big Open Linked Data

Tomasz J. Zlamaniec, Kuo-Ming Chao, and Nick Godwin

Original citation & hyperlink:
Zlamaniec, T.J., Chao, K.M. and Godwin, N., 2021. Workload-Aware Views Materialization for Big Open Linked Data. *Vietnam Journal of Computer Science*, *8*(02), pp.215-244.
https://dx.doi.org/10.1142/S2196888821500093

DOI 10.1142/S2196888821500093
ISSN 2196-8888
ESSN 2196-8896

Publisher: World Scientific

OPEN ACCESS
Vietnam Journal of Computer Science
Vol. 8, No. 2 (2021) 215–244

DOI: 10.1142/S2196888821500093

World Scientific
www.worldscientific.com

Workload-Aware Views Materialization for Big Open Linked Data

Tomasz J. Zlamaniec
OSP Control Systems
Ocado Technology, AL10 9NE, UK
t.zlamaniec@ocado.com

Kuo-Ming Chao* and Nick Godwin†
FTC, Coventry University
Coventry, CV1 2JH, UK
**k.chao@coventry.ac.uk*
†*angbt@btinternet.com*

Received 4 January 2020
Accepted 10 September 2020
Published 22 October 2020

It is a trend for the public organizations to digitalize and publish their large dataset as open linked data to the public users for queries and other applications for further utilizations. Different users' queries with various frequencies over time create different workload patterns to the servers which cannot guarantee the QoS during peak usages. Materialization is a well-known effective method to reduce peaks, but it is not used by semantic webs, due to frequently evolving schema. This research is able to estimate workloads based on previous queries, analyze and normalize their structures to materialize views, and map the queries to the views with populated data. By analyzing how access patterns of individual views contribute to the overall system workload, the proposed model aims at selection of candidates offering the highest reduction of the peak workload. Consequently, rather than optimizing all queries equally, a system using the new selection method can offer higher query throughput when it is the most needed, allowing for a higher number of concurrent users without compromising QoS during the peak usage. Finally, two case studies were used to evaluate the proposed method.

Keywords: Semantic web; QoS; view materialization; optimization strength.

1. Introduction

Increasingly volumes of open web data are provided by the public sector and other organizations in the EU and the UK to transparently publish the public service and scientific data they hold. This allows the public to access and reuse them to

stimulate a market and add value. The increasing usage has led to concerns over response times when large numbers of time critical queries are submitted concurrently, see Ref. 1. In particular, unevenness of requests to the datasets over time produce different workload patterns for the server with peak workloads at one time and idleness at other times. Several approaches such as: increasing computational powers with an elastic cloud; reducing storage IO accessing time; improving data access algorithms, and, optimizing datasets in Ref. 2, can balance peak workloads and improve efficiency in data retrieval. Materialization is frequently used in traditional relational database systems with restrictive structure, as a way to improve QoS. Unlike relational databases, the semantic data is described by an evolving schema. This creates new challenges, requiring views materialization that allows for open repositories of data to emerge. In the past, the possibility of introducing the views materialization process to semantic data was seen as unfeasible, due to rapidly evolving RDF schemas in Ref. 3. However, the emergence of a large number of semantic repositories such as WorldNet in Ref. 4, DBpedia in Ref. 5, Yago in Ref. 6 or European Union Open Data Portal see Ref. 1 has made the views materialization feasible, and resulted in the creation of new techniques that offer automated offline selection and materialization of views for SPARQL, e.g. in Refs. 7 and 8.

Various optimization techniques reduce the cost of executing individual requests or groups of queries. However, unlike the views materialization, most optimization techniques lack the ability to focus the optimization effort on a specific group of queries.

A new workload-aware views selection method is proposed to target the reduction of peak workload by exploiting the new characteristic of big semantic datasets. By selecting and materializing views based on their daily access patterns rather than the total number of affected requests, the proposed selection method focuses the materializing optimization effort. The paper is structured as follows.

Section 2 reviews the state of the arts in the view materialization. Section 3 presents the proposed workload-aware views selection method. In Sec. 4, there is further explanation of a candidate selection algorithm. Section 5 presents the proposed optimization system incorporating the proposed workload-aware views selection method and the candidate selection algorithm along with other components. A number of experiments on two case studies have been carried out to evaluate the proposed system, in Sec. 6, their results are reported, and the analysis on these results is presented in this section. Conclusion is drawn in the last section.

2. Literature Review

The views materialization problem for graph data was first introduced for XML and XQuery. The problem was targeted in Refs. 9–13. The author in Ref. 14 proposes a method for efficient query rewriting using multiple views simultaneously.

Although techniques proposed for relational and XQuery can be applied for SPARQL which is used in semantic webs, the modification of these techniques is not trivial as graph patterns used in SPARQL are less restrictive. Some of the techniques used for SPARQL limit the graph patterns to trees as found in XQuery (e.g. Chen and Chan's work in Ref. 15). Different attempts at adaptations of XQuery views materialization techniques to SPARQL were described in Refs. 16–18. The authors in Ref. 19 proposed a native method for analysis of SPARQL queries for the purpose of multi-query optimization.

Most materialization approaches for relational data aim at creating views that can be used as part of the solution in future queries. But many approaches to graph views, such as in Refs. 9, 11 and 15 for XQuery or in Ref. 14 for SPARQL, aim to create views that cover all queries, so that a rewritten query can return the same results through exclusively using the materialized views (without accessing the original data).

The problem of finding and merging view candidates is NP-Complete, and creates a potential bottleneck. As some of the candidates are less likely to bring benefit to the optimization, reduction of the initial number of candidates is desired. Typically, initially extracted candidates are filtered before merging, to reduce the complexity see Ref. 20. Rather than filtering individual queries, Le *et al.* in Ref. 19 used the Jaccard similarity coefficient as a heuristic that eliminates initial candidates not likely to be selected even after merging. The same principle was also applied by Roy *et al.* in Ref. 21, who proposed a set of heuristics based on dynamic programming to deal with nested sub-expressions. Tang *et al.* in Ref. 13 present a complete approach that provides efficient selection with a restriction on a single class of views.

While most merging techniques (e.g. by Le *et al.* in Ref. 19) search for the maximal common edge subgraphs in Ref. 22, Karanasos in Ref. 14 proposed an alternative bottom-up merging approach, that starts with minimal views that are combined to produce views answering particular queries.

Selection of candidate views is also an NP-complete problem and various approximation methods are used to reduce the amount of calculations needed. The selection is typically based on the benefit to cost ratio, where the optimization benefit is defined as the reduction in the total time of execution of all analyzed queries, while the cost is equal to the view's cardinality (and resulting increase in the dataset size after the view is materialized). A simplistic No-Cost model of cost estimation assumes that a view has acceptable cost if its structure (graph pattern) satisfies a complexity requirement in Ref. 23. However, as a view's cardinality can be estimated as a number of triples that would be materialized for the view, the problem of estimating the cost is reduced to the estimation of the number of triples that would be matched by the view. This estimation is one of the most basic functions of any modern optimizer in Ref. 24, and the cost estimation is a continuation of the What-If API Ref. 25.

Estimation of a view's optimization benefit is a separate problem. Trial materialization and execution of the analyzed queries is infeasible, even for a small number of views. Kaushik *et al.* in Ref. 26 used a cost model in which the query execution cost is modeled after the number of nodes in the query's graph pattern. This is intended to reflect the number of I/O operations needed to execute a query and does not include the possible growth in the number of intermediate results. Stocker *et al.* in Ref. 18 proposed the use of the traditional selectivity estimation (what-if) to estimate also the execution time of analyzed queries. Total estimated execution time under different views' configuration could be used to evaluate the benefits brought by each configuration. However, Liu *et al.* in Ref. 24 suggested that the model is not suitable for complex SPARQL queries and proposed a new execution cost model that is based on a SPARQL-specific set of statistics.

The selection of candidate views aims at maximizing the benefits (i.e. reduction of the total execution time for analyzed queries) while keeping the resulting set of views below the pre-specified size limit. An exhaustive search, finding all possible combinations of views and then removing the ones not suitable for the optimization, would not be feasible in Ref. 23. Instead, most of the recent approaches used a greedy strategy, where a single best view is selected in each iteration until an initially empty collection of results is full. Heuristics first used relational databases in Ref. 27 which have been successfully applied to both XQuery and SPARQL views in Ref. 15.

An alternative approach to views selection was proposed in Ref. 23, where the selection is performed before the merging process. Only the initially selected results are then used in merging, greatly reducing the total number of views analyzed in the selection.

Views materialization provides a viable approach to optimization for accessing semantic data with SPARQL. Although general optimization (aiming at reduction of the total execution time) can provide results for a high number of queries, it may not be the most optimal choice for dealing with the peak workload problem. An alternative approach to selection of candidate views could potentially reduce the peak workload problem, allowing for higher availability or better user experience.

3. Workload-Aware Selection of Candidate Views

A set of candidate views are generated from the existing queries in a database log, materialized, and stored in the repository for selection. The view selection is a process of optimizing queries by considering factors such as query workload and frequency that are used to maximize optimization benefit.

An example below shows three candidate views (named C_1, C_2, and C_3) and their data structures accessed by each of views are shown in Fig. 1 and their workload patterns in Fig. 2. Note that parts of C_1 and C_2 are overlapping.

As the data structure represented by candidate view C_2 is a subgraph of C_1, every request optimized by C_1 could be partially optimized by C_2, which means that the

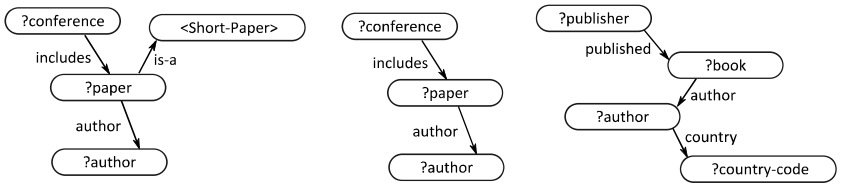

Fig. 1. Data structures of the example candidates (C_1 – left, C_2 – middle, C_3 – right).

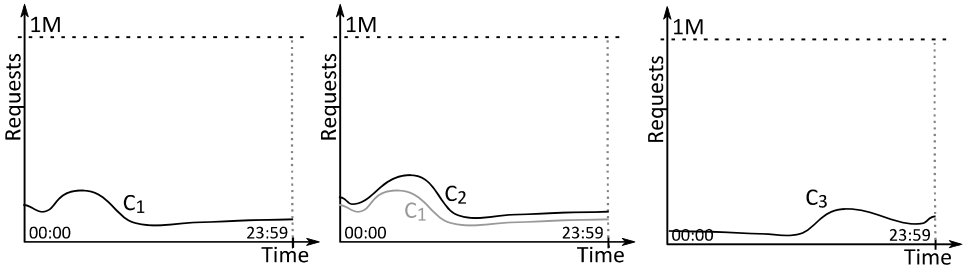

Fig. 2. Workload patterns of the candidate views (C1 – left, C2 – middle, C3 – right).

number of requests that could be optimized by C_2 is always higher. The overall system's workload is a combination of all individual access patterns, including access to data structures that cannot be optimized with views (e.g. from queries containing only a single predicate).

workload is defined as the number of queries executed during a specific time interval in the day in proportion to the highest number of queries executed at any time interval in the day Eq. (1).

$$\text{Workload}(t) = w(t) = \frac{\text{query count}(t)}{\max \textit{query count}}. \tag{1}$$

Each of the candidate views is characterized by a workload pattern. The patterns are constructed during the candidate extraction process, where each query request that would be affected by a view and the frequency of that view for the time interval containing the request.

The time interval size is one minute, i.e. t is the number of full minutes since midnight each day when the interval starts. The total number of queries per minute is counted not during a single day, but as a total count for the given time of day during the analyzed period. The cost of frequency is represented in Eq. (2).

$$\text{Frequency}(C) = \sum_{i=1}^{N} \text{workload}(\text{time}(R_i)), \tag{2}$$

where the R_i are the N query-requests related to the view C, and "workload()" returns values from zero (idle) to one (full capacity). As per definition of workload Eq. (1), the time is the time of day expressed as the number of full minutes since

midnight. The frequency for a candidate view is measured as the sum of the number of requests like weighting.

Materialization of a view requires the creation of new triples that have to be stored within the database. The cost of the materialization is defined as the number of triples that have to be materialized. As giving the absolute number of triples would not be informative, the cost is expressed as the percentage increase in the expected dataset size Eq. (3).

$$\text{Cost}(C) = \frac{\text{Number of materialized triples}(C)}{\text{Number of triples in the dataset}} \cdot 100\%. \tag{3}$$

In Eq. (3), the total number of triples in the dataset is the original dataset size. The triples materialized for previously selected views are not counted towards the dataset size.

To reflect the candidate's effect on the peak workload, the optimization benefit for the candidate C is defined as the peak workload reduction after the candidate is selected Eq. (4).

$$\text{Reduction}(C) = \left(1 - \frac{\text{peak}(S + C)}{\text{peak}(S)}\right) \cdot 100\%, \tag{4}$$

where "peak" returns the highest system workload after being optimized with already selected views (S) and after being optimized with already selected views and the current candidate ($S + C$). The peak workload is the highest workload value, and for differently optimized system workload, the peak can occur at different times of day.

Candidates with negative peak reduction provide no benefit to the previously selected set. However, the calculated value depends on the previously selected views (S) and needs to be recalculated after another view is selected. Therefore, these candidates are not rejected from the selection process.

The workload itself is a function of time, i.e. for any given time of day the system workload value is equal to the number of requests during that time. When the optimized system workload is being estimated, the effect of each materialized view is subtracted. The optimized workload is defined as:

$$ow(t, S) = w(t) - \sum_{V \in S} (f(V, t) - f(V, t) \cdot o(V)), \tag{5}$$

where S is the collection of materialized views V used in optimization, $f(V, t)$ is the view's frequency, and $o(V)$ is the strength of the optimization (with 0 indicating full optimization, and 1 indicating no effect) in Eq. (5).

The workload at the time $\mathbf{t}$ optimized with a collection of views $\mathbf{S}$ is equal to the overall system workload at that time $\mathbf{w(t)}$ reduced by a sum of the optimization effects from each of the materialized views. The optimization effect of each view is

proportional to the product of the view's frequency at the time and its effect on a single query.

Optimization strength: The optimized workload is calculated with use of the information about the overall system workload, the frequency of candidate views, and the effect that each view materialization has on affected queries.

While the overall system workload and views frequency are calculated during the log analysis, the optimization strength that a view has on a query has to be estimated.

If constant parameters (such as hosting performance) are ignored, the execution time of a query depends mostly on the query's complexity and selectivity. Due to the way in which data is organized into indices, edges in a star configuration (coming out of one node) can be evaluated efficiently, while edges forming a path (chained nodes) are inefficient.

Evaluation of a path expression requires the edges to be evaluated in a sequence, with each edge generating possibly invalid intermediate results. With the assumption that the average selectivity of a predicate with a known subject is two, the number of intermediate results for graph pattern, g, can grow to $2^{|g|}$, as the example in Fig. 3 shows.

In the example, execution of the first edge returns multiple possible results (books). Each of these is used in matching of the second edge and produces one or more additional intermediate results for every possible book. The query result is found with the execution of the fourth pattern after which only the correct results (matching the last edge) are accepted.

Execution of the first edge returns multiple possible results. Each of these is used in matching of the second edge and produces one or more additional intermediate results for every possible outcome. The query result is found with the execution of the fourth pattern after which only the correct results (matching the last edge) are accepted.

With this model, removal of N edges from a path containing $|g|$ edges results in reduction of query complexity by a factor of 2^{-N+1}. As high estimation precision is

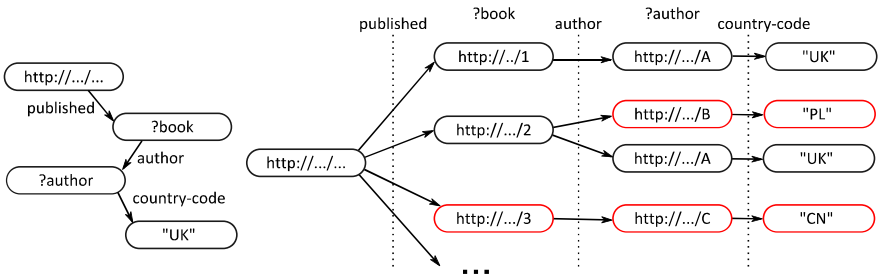

Fig. 3. Example of how the number of intermediate results grows with matching of each next edge of a graph pattern.

not required (if the same method is used for all candidate views), the estimation model for the optimization strength can be simplified.

$$o(V) = 1 - \frac{2}{2^{|V|}}, \tag{6}$$

where $|V|$ is the number of edges in the materialized view in Eq. (6). Higher value means that the view provides higher optimization strength.

The equation defining the optimized workload defined in Eq. (5) can be modified to include this optimization model.

Knowing that:

$$\begin{aligned}\frac{2}{2^{|V|}} \cdot f(V,t) &= 2 \cdot \frac{1}{2^{|V|}} \cdot f(V,t) = 2 \cdot 2^{-|V|} \cdot f(V,t) \\ &= f(V,t) \cdot 2^{-|V|+1}.\end{aligned}$$

The final equation (Eq. 7) can be written as:

$$ow(t,S) = w(t) - \sum_{V \in S} (f(V,t) \cdot 2^{-|V|+1}). \tag{7}$$

This allows estimation of changes in the workload after a candidate or collection of candidate views is selected. In order to accommodate views, a time restriction needs to be added to Eq. (4). With a focus window defined as a set containing a list of time intervals (minutes) that should be included in the calculation of Eq. (8):

$$\textbf{\textit{IF}}\ t \in \text{Focus\_Window}\ \textbf{THEN}\ \text{include time}\ t. \tag{8}$$

Equation (4) is modified to Eq. (9).

$$\text{Reduction}(C) = \left(1 - \frac{\text{peak}(S + C, \text{Focus\_Window})}{\text{peak}(S, \text{Focus\_Window})}\right) \cdot 100\%, \tag{9}$$

where the "peak" is defined as the highest workload value within the focus window:

$$\text{peak}(S, \text{Focus}_{\text{Window}}) \rightarrow \max\{ow(t,S) : t \in \text{Focus}_{\text{Window}}\}. \tag{10}$$

If the optimization focus is not required during a specific time, then the focus window spans over the period of 24 h (i.e. all values are considered). An example of the benefit estimation with and without the optimization focus is shown in Fig. 4.

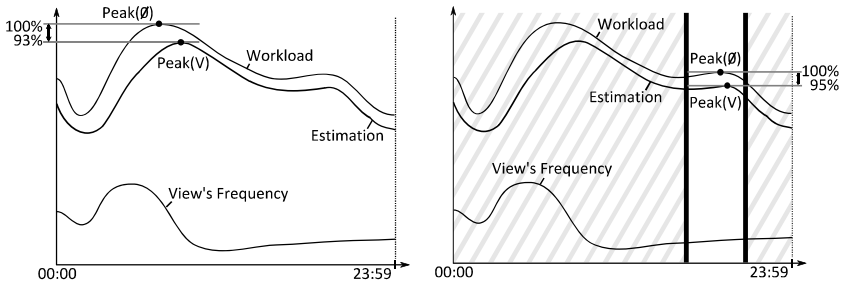

Fig. 4. Optimization benefit calculated without (left) and with a single focus window (right).

4. Candidates Selection Heuristics

The selection of best candidates is a variation of a Knapsack problem which aims at selecting items of the highest value (peak workload reduction) with restriction on the total cost (size of materialized data) in a complex and large search space. Searching for an optimal solution for the Knapsack problem is known to be NP-complete. Therefore, it is most likely infeasible for a high number of candidates. Instead, the selection can be performed with the use of greedy heuristics.

At the input, the selection algorithm receives a set of candidate views, each characterized by Candidate View's Cost: *Cost(C)* and Candidate View's Optimization Benefit: *Reduction(C)*.

The greedy selection method reorders candidates based on the cost ratio, so that the highest ratio candidates are placed first, so that:

$$\frac{\text{Reduction}(C_i)}{\text{Cost}(C_i)} \geq \frac{\text{Reduction}(C_{i+1})}{\text{Cost}(C_{i+1})}. \tag{11}$$

In every step, the candidate with the highest ratio is selected until the total cost limit is reached. This solution is considered sufficiently accurate for most inputs, but in a worst-case scenario the accuracy is low.

To further increase the accuracy, the algorithm is extended with an additional step. After completing the selection, the same algorithm is repeated with the initial solution set containing the candidate view with a higher optimization value (rather than the value to cost ratio). This produces an alternative set of selected candidates, and the set offering a higher total reduction of peak workload is returned.

With the assumption that invalid candidates (with prohibitive cost) are rejected prior to the selection, the adopted heuristics is known to guarantee that in the worst case the selection is at least 50% as efficient as the optimal solution in Ref. 28. The selection algorithm is shown in Fig. 5 (the initial rejection of weak candidates and the candidate's cost estimations are not included). In every iteration of the main loop (lines 4–18), the algorithm selects the candidate view with the highest ratio between the expected optimization benefit and the materialization cost.

While the cost of each candidate remains constant, the optimization benefit depends on already selected views. Therefore, the benefit of remaining candidates has to be recalculated for all remaining candidates, before the selection can take place (lines 5–8). Afterwards, the candidates are ordered according to the optimization to cost ratio (line 9) and the first valid candidate is selected (lines 10–17). To handle the worst-case scenario, the algorithm creates an alternative selection result (line 22) and initially selects the candidate with the highest estimated optimization, regardless of the materialization costs (lines 23–26). The original selection algorithm is then called recursively (27) to select additional candidate views until the limit is reached. After both the original and the alternative selection result are produced, the algorithm returns the result offering higher optimization benefit (line 29).

```
1.  V – Remaining candidates
2.  S – Selected candidates
3.
4.  WHILE V is not empty REPEAT
5.      FOR-EACH remaining candidate C belonging to V DO
6.          Remove the candidate C if its size is higher than the
    remaining limit
7.          Estimate new optimisation benefit for C, assuming S are
    already materialised                    Eq.( 4, 8)
8.      END-FOR
9.      Sort V in non-decreasing order of the optimisation benefit to the cost
    ratio  Eq. (7)
10.     WHILE V is not empty and no candidate was selected yet DO
11.         SET C = first remaining candidate from V
12.         Remove C from V
13.         IF Selecting C does not violate the cost limit THEN
14.             Select C by adding it to S
15.             Break the while loop
16.         END-IF
17.     END-WHILE
18. END-WHILE
19.
20. --- Additional check for the worst-case
21.
22. SET S' = new empty set for storing alternative selection
23. FIND Remaining candidate C with the highest optimisation estimate
24.     Remove C from V
25.     Select C by adding it to S'
26. END-FIND
27. CALL the algorithm recursively starting with the set S' instead of S
28.
29. RETURN set solution S or S' with the higher total optimisation benefit
```

Fig. 5. Pseudocode for the selection algorithm.

The heuristic approach does not guarantee that the selection result is optimal. The greedy heuristic selects candidates according to the ratio between the optimization benefit and the materialization cost of each candidate view. It is possible that a low-cost candidate selected due to its relatively high benefit to cost ratio will prevent a more costly candidate from being selected. The following example illustrates the problem. The total size available for all materialized view is equal to 100%. Candidate 1 offers 20% reduction of the peak workload and requires 20% of the total permitted size. Its benefit-to-cost ratio is 1. Candidate 2 offers 45% reduction of the peak workload and requires 90% of the total permitted size. Its benefit to cost ratio is 0.5.

Having to decide between the two candidates, the greedy heuristic selects the first candidate and terminates, as the remaining candidate cannot fit in the allowed space. The resulting selection is therefore not optimal, as the overall peak reduction would be higher if only the second candidate was selected. With higher number of

candidates, the same problem occurs if multiple high ratio candidates are selected in favor of a single candidate that offers higher optimization benefit than all of the selected candidates together.

In order to improve the effectiveness of the selection, the proposed algorithm produces an alternative solution. In the alternative solution, the first selection step is not based on the benefit to cost ratio but on the optimization benefit alone. The initial and the alternative solution are compared, and the solution with higher optimization benefit is selected.

By producing the alternative solution, the algorithm recovers from the worst-case scenario. This method of improving the accuracy in solving a general Knapsack problem is known to guarantee that the total value of selected items is no lower than 50% of the optimal solution. A formal proof is provided by Kilpelainen in Ref. 28.

5. System Design and Implementations

The workload aware view selection and candidate selection heuristics are the core components of the optimization system. The fundamental concept behind the system is the ability to analyze past queries in order to estimate which data structures (views) are most likely to be required by future queries. Based on the past queries, the system proposes and materializes new views that can be used as partial results for answering future queries. Rather than being integral part of a SPARQL engine, the system is designed to serve in the proxy layer between users and a SPARQL endpoint. Being enclosed in a proxy, the optimization system can be added to existing engines without increasing the complexity of the engine. Furthermore, other optimization methods previously incorporated into the SPARQL engine are preserved.

The major tasks realized by the system are:

(1) Extracting Frequent Graph Patterns from the log of previous queries. Past queries are analyzed in order to extract and analyze frequently accessed data structures. Individual access (workload) patterns are generated separately for each structure.
(2) Proposing new candidate views that optimize the found patterns. A number of alternative graph patterns are proposed to replace complex data structures (views) with simpler structures that require less joins.
(3) Selecting the views configuration that offers the highest optimization. With limited space, it would be infeasible to materialize all views. Instead, a set of views is selected based on the potential benefits expected from materialization of each data for view.
(4) Modifying incoming queries to make use the newly materialized data. An incoming query does not get any optimization benefit from the materialized views unless the query's structure is modified to access the new data. If possible, any incoming query is transparently altered to make use of the materialized data while ensuring the same results.

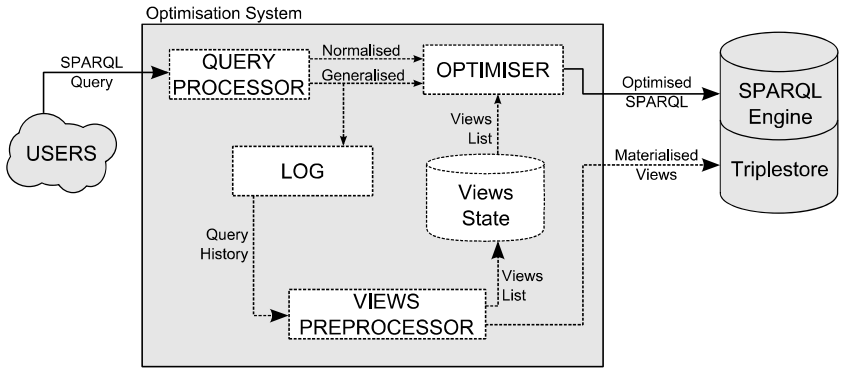

Fig. 6. Modules composing the framework (dashed lines show background operations).

In order to perform the aforementioned tasks, a typical triples retrieval system has to be extended with new modules, as shown in Fig. 6.

While a SPARQL Engine and a Triple Store are sufficient to answer a valid SPARQL query, the implemented optimized system introduces additional modules. A views preprocessor will perform analysis of the past queries to materialize new views. A query processor parses the incoming queries and an optimizer compares an incoming query's structure with previously materialized views. Two additional elements in the system the Queries Log, and Views' State table contain information about currently available views in order to provide the two main functions: preparation of views and querying. The view preparation involves analyzing past queries to propose new candidate views, selecting a subset of candidates offering the best potential optimization; materializing views, and, inserting new triples into the dataset. The querying includes: adding an incoming query to the log; finding materialized views matching the query; altering the query if possible, and, executing the query. More details of the implementation can be found in Ref. 29. In the next section, the implemented system is evaluated with the different datasets and parameters to comprehend the effectiveness of the proposed optimizer.

6. Evaluations

The approach proposed in this research aims at optimizing the peak workload for servers hosting semantic data accessible with SPARQL. The introduced optimization method is based on two assumptions: (I) Materializing SPARQL views allows for the more efficient execution of query, and (II) Different data structures within the database are characterized by different workload patterns.

The first assumption comes from the analysis of the indices structure in native triple stores and the way in which long queries are executed. The analysis has shown that substituting long path expressions with star expressions can reduce the number of I/O operations, thus increase the performance.

The second assumption comes from the fact that open repositories can combine information from different domains. Therefore, one must expect to manage data from different domains that exhibit different access patterns depending on the daily changes in user interests.

Based on these assumptions, a hypothesis is proposed:

Inclusion of the workload patterns of individual SPARQL views in the views selection process allows the peak workload reduction to be higher than the reduction offered by a frequency-based approach.

This section presents an in-depth evaluation of this hypothesis and the underlying assumptions. The evaluation is composed of two major phases. The first phase is conducted with controlled data and evaluates the effect of views materialization in SPARQL. The second phase evaluates the proposed selection method with the use of the implemented prototype. In this phase, the evaluation is conducted with a publicly available dataset containing general knowledge, and with real-world queries.

The first test evaluates the prototype with a subset of data generated for the GREENet project which includes an information system to manage large amount of the waste of electronic and electrical equipments and their tracking data. The used data are characterized by relatively simple structure and a limited number of unique queries. The test is intended to confirm the fundamental assumptions behind the proposed idea, i.e. to verify if view materialization allows for more efficient query execution, and if the eventual benefits outweigh the negative effects of increased data size.

The second phase of the experiment involves data from DBpedia (version 3.9), a publicly available dataset containing general semantic knowledge.[a] The dataset used in the experiments contains approximately 4 million entities described with 470 million triples. In addition, OpenLink Software in Ref. 30 offers real-world query logs for the dataset.[b] The DBpedia dataset and queries are traditionally used for the evaluation of a SPARQL engine's effectiveness.

6.1. *The testing environment*

All tests performed during the experimental evaluations were executed using the configuration and techniques specified in this section.

The evaluation of the proposed framework is performed with a specially implemented prototype system. The basis of the prototype is a state-of-the-art native triple store, Jena TDB, version 2.7.4. The queries are parsed using version 2.6.9 of Sesame.

For each of the tests involving query execution, the prototype is tested in three modes.

- Direct execution of queries without optimization. The queries are executed on the base dataset as a baseline

[a] http://wiki.dbpedia.org/Downloads39.
[b] ftp://download.openlinksw.com/support/dbpedia/.

- Execution of not-optimized queries on a dataset containing materialized data. The original queries are executed on the dataset containing the materialized views to measure the extra overhead resulting from the extended dataset size
- Execution of optimized queries. Queries for which a materialized view is available are optimized and executed together with remaining queries to measure the optimization effect on the workload.

Correctness of the implementation was tested prior to the evaluation by executing pairs of optimized and not-optimized queries and ensuring that both queries return the same results. The results' ordering is irrelevant for a SPARQL query unless the ORDER BY closure is used.

The evaluation was performed on a single server controlled by Windows Server 2008 R2 (with memory-mapped files enabled). The server contains 12 GB of DDR3 RAM and Intel i5 2500 k CPU at 3.30 GHz. The data were stored on two hard drives connected in RAID 1 configuration. The storage offers read speed of 240 MB/s and average random access time of 4 ms.

The query throughput is measured by executing all queries from a given set in a sequence. The throughput is defined as the total number of queries in the set, divided by the total execution time (in seconds).

Short experiments executed on a Java Virtual Machine can be affected by the unpredictable execution of the Garbage Collector (GC). Typically, this difficulty is managed with the test being repeated multiple times and the worst result being rejected.[c] However, this procedure is not normally observed in an evaluation involving the execution of a very high number of queries, as the effects of the GC are negligible. Instead, each test involving the execution of queries is repeated three times, and the average result is reported.

Every test involving query execution is preceded by the cleaning of any caching mechanisms that might affect the test outcome (including the memory-files mapping provided by the operating system). The cache invalidation is followed by a warm-up phase, during which a set of 1,000 randomly selected queries is executed.

6.2. *Evaluation in controlled environment*

The initial evaluation of the prototype system is performed in a controlled environment. This evaluation confirms whether the materialized views can provide enough benefit to SPARQL query optimization to outweigh the negative effect of increased data size.

The evaluation is performed with use of the queries and a subset of the data generated for the GREENet project. The test dataset in this test is composed of 120 k entities described by approximately 1 million triples.[d] This initial evaluation includes

[c] The GC can suspend the execution of the test program for several milliseconds. GC occurring during a slow operation can increase its time considerably.
[d] The exact number of triples in the dataset is 1,035,301.

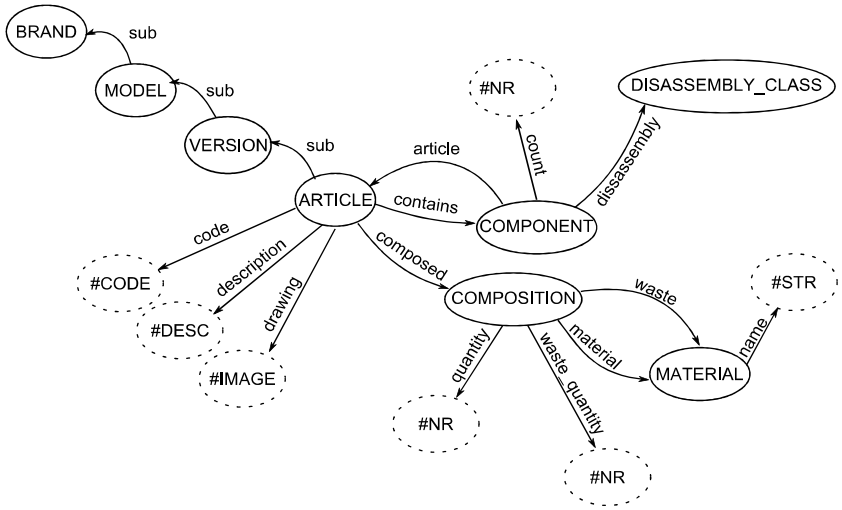

Fig. 7. A subset of the GREENet schema accessed by the evaluated queries.

analysis and optimization of the top 50 query structures used to query the schema (Fig. 7).

Figure 8 is an example of a test query used on GREENet dataset.

The query retrieves all articles containing soldered metal components and the name of the specific metal used. The experiment involves a set of 68,000 queries that access 50 unique data structures. All of the used data structures (and queries) can be optimized with views materialization.

6.2.1. *Experiment and results from GREENet*

This experiment is designed to verify whether the view materialization allows for more efficient query execution, and if the benefits (if any) outweigh the negative effects resulting from increased data size.

The test queries are grouped into 50 sets where each set can be optimized with a different materialized view. The groups are then ordered according to the number of queries (in decreasing order). The threshold values are rounded to the nearest 50.

```
SELECT ?art ?matname WHERE
{
        ?art a gn:Article .
        ?art gn:contains ?component .

        ?component      gn:assembly
        <http://greenet.com/data/assembly#solder> .
        ?component      gn:contains     ?material .

        ?material gn:type <http://greenet.com/data/type#metal>
        ?material gn:label ?matname
}
```

Fig. 8. An example of a GREENet query.

The test procedure is as follows: (1) Load the original dataset; (2) Execute all original test queries; (3) Select and materialize the top N most frequent views; (4) Execute all the original queries; (5) Execute all optimized queries; (6) Execute a combination of optimized (if possible) and not optimized (otherwise) queries.

Each execution step is performed as specified before in Sec. 6.1 (with cleanup, and warm-up). Starting with no views being optimized, each test iteration adds a higher number of views to be materialized (always selecting the most frequent views), until all views are materialized in the final iteration.

For each of the tests, the measured characteristics are: query throughput for the optimized queries; query throughput for not optimized queries; overall query throughput.

The experiment results are grouped into three categories, which are: (1) materialized views effect on the overall throughput optimization; (2) increase of the dataset size and its effects; and, (3) combined changes in overall, optimized, and not optimized query throughput.

6.2.2. *Results: Optimization effect*

Structures from the most frequent to the Nth most frequently accessed e.g. the most frequent data structure is accessed by 4.7% of queries, while data structures from the most frequent to the 20th most frequent are accessed by 62% of queries.

The data structures' access frequency has power-law distribution, i.e. frequency decays exponentially. As shown in Fig. 9, the 15 most frequent data structures are accessed by approximately 52% of all test queries.

Figure 10 shows the effect that view's materialization has on the throughput for the affected queries. For clarity, values are rounded to the nearest 50. Percentage values are rounded to the first two meaningful digits.

"Affected queries" is the percentage of queries optimized by the selected view e.g. with the top five views materialized, 23% of the executed queries were optimized by these views while the remaining 77% were executed without optimization. **"Ex. Time"** is the total time needed to execute all of the queries.

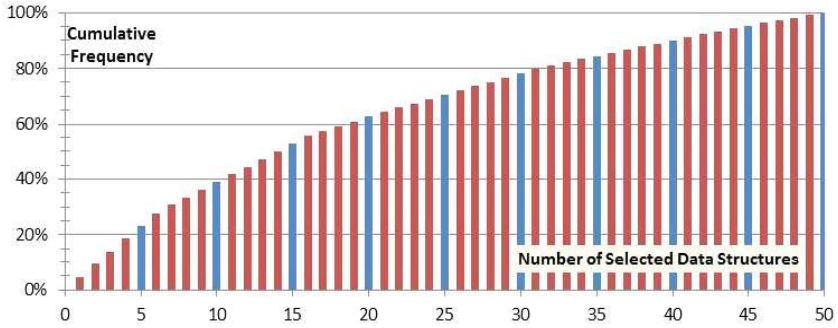

Fig. 9. Cumulative number of queries accessing the top 50 most frequent data structures.

Selected views	0	5	10	15	20	25	30	40	50
Affected queries	0%	23%	38%	52%	62%	70%	82%	93%	100%
Ex. Time [sec]	34.3	27.8	22.7	18.6	17.2	16.0	15.3	14.6	14.0
Throughput	2,000	2,450	3,000	3,650	3,950	4,250	4,450	4,670	4,850
Relative	100%	124%	152%	184%	199%	215%	225%	236%	245%
Increase	-	24%	22%	22%	8.2%	7.6%	4.7%	5.0%	3.9%

Fig. 10. Change in the overall query throughput for different number of materialized views.

"Relative" is the relative throughput as compared to the original data and queries (with no views selected). **"Increase"** reflects how the throughput has increased with the selection of an additional five views for materialization.

Figure 11 shows the relation between the percentage of queries that have been optimized and the overall query throughput. The percentage value represents the fraction of queries that are optimized with use of the selected views.

The materialization of each additional view added new triples to the dataset resulting in dataset's growth. The table in Fig. 12 shows the relative dataset changes (with different number of views being materialized).

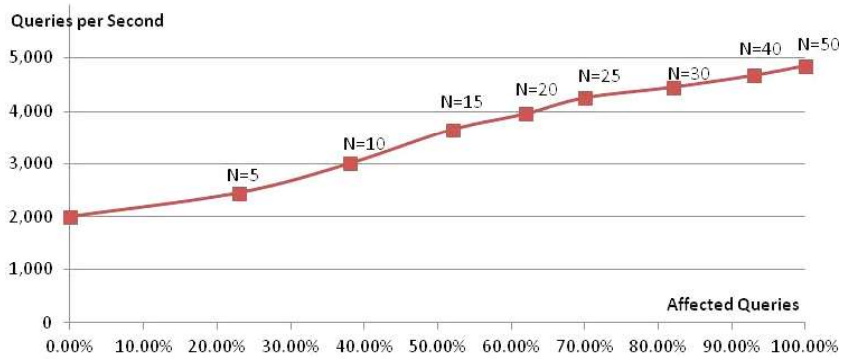

Fig. 11. Overall throughput change in relation to the percentage of optimized queries for a different number of optimised views N.

Views (N)	0	5	10	15	20	25	30	40	50
Added triples	-	41k	59k	87k	110k	139k	164k	212k	260k
Added size	-	**4.1%**	**5.9%**	**8.7%**	**11.0%**	**13.9%**	**16.4%**	**21.2%**	**26.0%**
Not-Opt.	2,000	1,95	1,95	1,75	1,650	1,700	1,850	1,700	1,750
Throughput	100%	98%	98%	87%	83%	85%	93%	85%	87%

Fig. 12. Increase in dataset size relative to the number of selected views N.

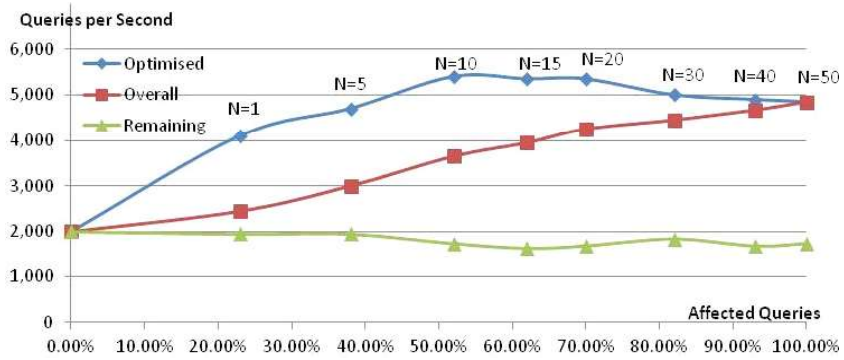

Fig. 13. Combination of optimized, not-optimised, and overall query throughput for a different number of materialized views N.

"Added triples" is the number of additional triples that were added to the dataset. **"Added size"** is the percentage increase of the storage space used to host the data. **"Not-Opt. Throughput"** is the query throughput for the all queries without performing the optimization, expressed as the absolute value and percentage change (compared to the original dataset).

The graph in Fig. 13 shows how the overall query throughput (for both optimized and not-optimized queries) changes as more views are selected for materialization. The overall query throughput after the optimization is the combination of the throughput queries optimized with use of materialized views, and queries that were not optimized. "Remaining" is the throughput for queries that were not optimized by any of the materialized views, "Optimized" is the throughput for optimized queries, and "Overall" is the throughput for all queries in the dataset.

The top line represents only the throughput for the queries that were optimized, while the bottom line represents the throughput for queries that were not optimized.

6.2.3. *Results summary*

This phase of the experiment was intended to test the initial assumptions and compare the positive and negative effect of SPARQL Views Materialization. Figure 14 shows that optimization of all SPARQL queries used in the experiment increases the throughput to 243% on the starting value.

The results show that the optimization strength varies with the number of queries optimized, which is explained by different optimization strength for different data

Views (N)	0	5	10	15	20	25	30	40	50
Query/s	2,000	4,100	4,700	5,400	5,350	5,350	5,000	4,900	4,850
Execution Time	34.3	27.8	22.7	18.6	17.2	16.0	15.3	14.6	14.0
Throughput	100%	205%	235%	270%	268%	268%	250%	245%	243%

Fig. 14. Views materialization effect on optimized queries.

Views (N)	0	5	10	15	20	25	30	40	50
Dataset size	0.0%	4.1%	5.9%	8.7%	11%	13.9%	16.4%	21.2%	26%
Query/s	2,000	1,950	1,950	1,750	1,650	1,700	1,850	1,700	1,750
Throughput	100%	98%	98%	88%	83%	85%	93%	85%	88%

Fig. 15. Decrease of query throughput for queries not optimized by views.

structures materialized in each set. In addition, although the overall throughput increases, the throughput for optimized queries begins to decrease after for the sets with 40 and 50 top query structures materialized. This can be explained by low number of queries benefiting from the optimization of the additional views combined with the negative effect that increased dataset size has on the query execution time (Fig. 15).

As expected, the increase of the dataset size caused by materialization of views has negative effect on the queries that cannot be optimized by any of the available views. The penalty on total execution time for the not-optimized queries reaches the maximum of 15% and approaches 12% when the dataset size is increased by 26%. This result is consistent with the results obtained Morsey *et al.* in Ref. 31 where, for various databases, the performance decrease caused by increase of the dataset size degrades by approximately 20–30% when the original dataset is doubled in size in Ref. 31.

While the increased data size has negative effect on some queries, the data in Fig. 16 shows that the overall query threshold (when executing both optimized and not-optimized queries) is increasing with materialization of every additional view.

However, the data shows that every next set of optimized data structures is less effective than the previous, i.e. the ratio between the optimization strength and the total cost of materialized views decreases. This allows a conclusion, that with a higher number of unique data structures, optimization of additional data structures above certain point can give no further benefit.

With the assumption that real-world queries access a much higher number of unique data structures, the view materialization is only beneficial when limited to the top structures, thus requiring a selection mechanism.

Views (N)	0	5	10	15	20	25	30	40	50
Dataset size	100%	104%	106%	109%	111%	114%	116%	121%	126%
Query/s	2,000	2,450	3,000	3,650	3,950	4,250	4,450	4,670	4,850
Throughput	100%	123%	150%	183%	198%	213%	223%	234%	243%
Change		+23%	+27%	+33%	+15%	+15%	+10%	+11%	+9%
Benefit/Cost Ratio	-	0.85	0.71	0.59	0.56	0.53	0.52	0.52	0.52

Fig. 16. Overall effect of views materialization. The benefit/cost ratio is the ratio between the increased throughput and the dataset size.

The results presented in this section were collected in the first phase of the experiment. The next section presents evaluation of the prototype system with real-world data. A discussion regarding the relation between the both phases of the experiment and the established conclusions are presented in the discussion section.

6.3. *Evaluation with real-world data*

The second part of the evaluation tests the main hypothesis, which states, that placing the views selection focus on individual workload patterns allows a reduction in the peak workload. The experiments within this phase are conducted with a publicly available dataset containing general knowledge, and with real-world queries.

The evaluation is performed with use of real-world dataset DBPedia that contains semantic data extracted from the Wikipedia articles and links it to other semantic datasets. The data contains over 4 million resources described with over 470 million triples (DBPedia 2014).

The queries used in the evaluation were published by OpenLink Software (2011). The published query log was collected over two weeks on a publicly available server. The log includes approximately 10.2 million queries. As a preparation for the evaluation, an attempt was made to parse all queries. Queries that could not be parsed (due to incorrect SPARQL) and queries containing elements not supported by the prototype (approximately 4.5% of all queries — see Sec. 5.8) were removed, leaving approximately 9.4 million of queries.

The DBpedia dataset and queries are traditionally used for the evaluation of a SPARQL engine's effectiveness. The publicly available query log collected by the server is used as the source of real-world queries.

The DBPedia dataset and queries are commonly used to evaluate SPARQL databases and create a base for several evaluation frameworks such as the DBPedia SPARQL Benchmark in Ref. 31.

During the evaluation, the dataset is loaded into the database and the published queries are imported into the log of the proposed framework. Afterwards, a series of tests is executed to measure the effects of the workload-aware views selection. The evaluation steps are:

(1) Load the DBPedia dataset into the Jena TDB Database
(2) Import the DBPedia queries into the Query Log of the framework's prototype
    - Parse and attempt to execute every query to ensure that only valid SPARQL queries are accepted. Queries that cannot be parsed and queries taking more than 5 s to execute are rejected.
(3) Use the prototype framework to propose candidate views
(4) Reject weak candidates
(5) Calculate the total materialization cost of all remaining candidate views
(6) Calculate the total system's workload

(7) Execute the tests of selection algorithms

- Start with the cost limit equal to 2% of the total and repeat the tests until reaching 100%. For every iteration, accept the result only if it is different from the result of the previous iteration.
  In each iteration:
- Select a set of views using the workload-aware selection method
- Select a set of views with use of the same heuristic as the workload-aware selection, but use the total frequency of a candidate view as the optimization benefit indicator (instead of the percentage reduction in peak workload)

During the experiment, the selection heuristic was executed twice for every cost limit. The workload-aware benefit model was used during the first run (Sec. 3), and the frequency-based benefit model was used during the second run. In the workload-aware model, the optimization benefit for a candidate C depends on the change in peak workload after the candidate selected see Eq. (4).

In the frequency-based mode, the benefit of an optimization is modeled using the total frequency of all data structures affected by the view, see Eq. (12)

$$\text{Freq. benefit}(C) = \sum_{t=0:00}^{23:59} \text{frequency}(C, t). \tag{12}$$

The simplest queries containing only one statement pattern are not included in the evaluation.

6.3.1. *The measured workload and access patterns of top candidate views*

To show an example of how the frequency of different candidate views matches the overall system workload, the access patterns from two of the top 10 candidate views and the overall workloads are shown in Fig. 17 (two diagrams). The access pattern

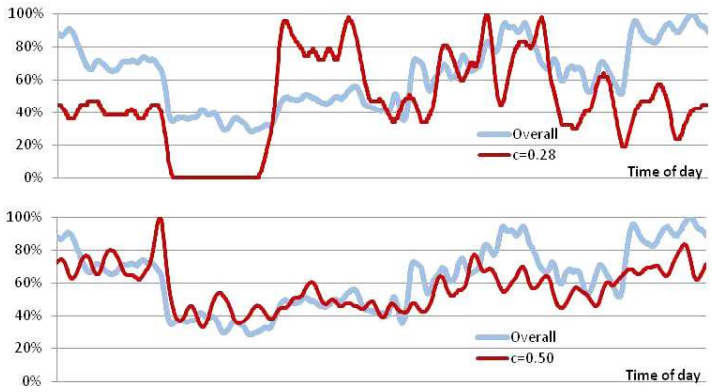

Fig. 17. Access pattern for one of the most frequent candidates compared with the overall system's workload, the c value is the correlation between the two.

Correlation Range C				
From	To	Candidates	Category	Candidates
0.0	0.09	30.9%	Zero	30.9%
0.1	0.19	20.9%	Weak	31.8%
0.2	0.29	10.9%	Weak	
0.3	0.39	19.1%	Moderate	
0.4	0.49	3.6%	Moderate	25.5%
0.5	0.59	2.7%	Moderate	
0.6	0.69	0.9%	Strong	
0.7	0.79	1.8%	Strong	2.7%
0.8	0.89	0.0%	Strong	
0.9	1.00	0.0%	Perfect	0.0%

Fig. 18. Correlation between the overall system workload and access patterns of individual candidates.

graph for a candidate shows how often the data structure targeted by the candidate view is accessed at any given time (expressed as the percentage of the maximum).

The analysis of the correlation for candidates used in the selection (i.e. not rejected as weak candidates) is shown in Fig. 18.

While access patterns of all candidate views contribute to the overall workload, the correlation data shows that different candidates contribute differently to the shape of the workload curve. The higher the colloration value c, the stronger contribution to the overall workload.

Different patterns contribute differently to the overall system workload, thus confirming the second initial assumption stating, that different data structures within the database are characterized by different workload patterns.

6.4. *Effect on the optimization on peak*

6.5. *Workload in Fig. 19 presents the highest modified system workload with different limits for the total cost of materialized views. The cost limit varies from 0% to 160% of the original size of the dataset; thus, the modified dataset size shown in the graph is in range between 100% and 260%*

The results show that in a cost range of up to 60% of the original dataset size, the workload-aware selection offers nearly linear reduction of the highest workload. Furthermore, the reduction offered with use of the workload-aware model is higher than the reduction offered by the frequency-based model of estimating the benefit of a candidate view. However, as the total frequency of individual views is not

considered as the selection factor, the overall number of optimized queries is lower (Fig. 20).

To better show the tradeoff between the positive effects (i.e. higher reduction of the peak workload) and the negative effects (lower total number of affected queries), the graph in Fig. 21 shows the differences in effect obtained with workload-aware and frequency-based selection.

The difference in peak decrease is calculated as the difference between the percentage peak decrease for workload-aware and frequency-based selection in relation to the lower number.

$$\text{Peak Reduction} = \frac{\text{Workload Aware Reduction} - \text{Frequency Based Reduction}}{\min(\text{Workload Aware Reduction}, \text{Frequency Based Reduction})}.$$

The relative difference in the number of affected requests is calculated as:

$$\text{Optimized Requests} = \frac{\text{Workload Aware Frequency} - \text{Frequency Based Frequency}}{\min(\text{Workload Aware Frequency}, \text{Frequency Based Frequency})}.$$

The data shown in Figs. 19–21 is presented in the summary table in Fig. 22.

The data shows that the peak workload can be reduced to 62% of the original value when the size of the dataset is doubled (with the materialization cost limit set to 100% of the original dataset size). By comparison with 50% size increase allowed, the peak workload is reduced to 68% which leads to conclusion that the efficiency decreases for the higher limit. This tendency is shown in Fig. 23.

When comparing the overall optimization effect, a quality typically used when evaluating the effects of an optimization, the workload-aware selection offers lower benefit than the standard frequency based approach. The choice of a selection method is therefore a tradeoff between a stronger peak reduction and the reduction in the execution time of all queries.

The evaluation data for the materialization cost limit set to 10% of the original dataset size differs from the majority of the datapoints. At that point, both the

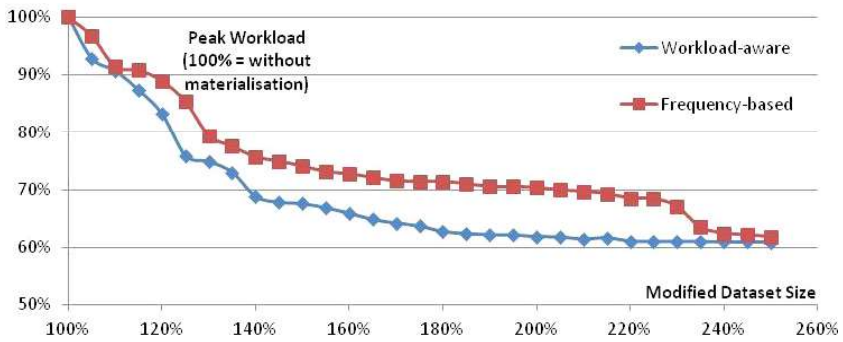

Fig. 19. Highest modified system workload for different increases in dataset size.

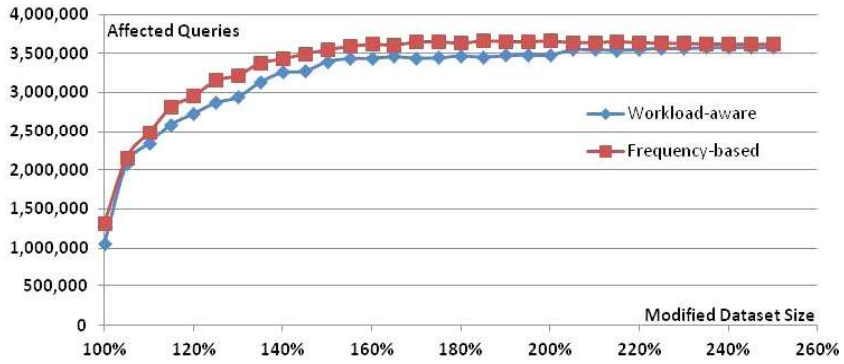

Fig. 20. Total number of optimized request for different increase in dataset size.

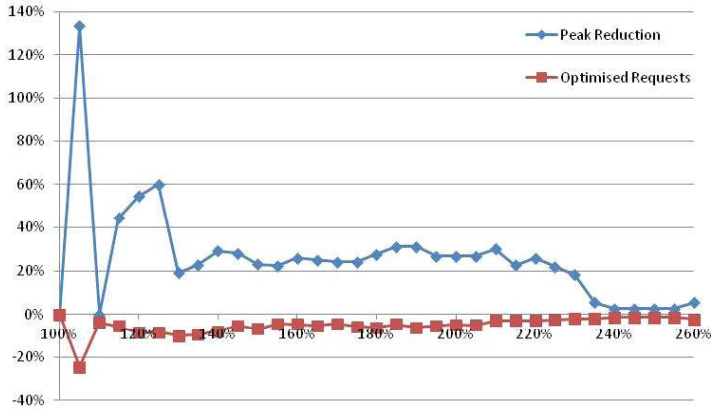

Fig. 21. Relative difference in the effects of using the workload-aware method and the frequency-based selection method.

Dataset Size	105%	110%	120%	130%	140%	150%	160%	180%	200%	260%
Peak modified workload										
W-aware	93%	91%	83%	75%	69%	68%	66%	63%	62%	61%
F-based	97%	91%	89%	79%	76%	74%	73%	71%	70%	63%
Total optimised requests										
W-aware	1,060	2,080	2,600	2,870	3,140	3,280	3,440	3,450	3,490	3,580
F-based	1,320	2,160	2,820	3,160	3,390	3,500	3,600	3,660	3,660	3,660
Relative difference										
Peak reduction	133%	0%	55%	19%	29%	23%	26%	28%	27%	5%
Opt. Requests	-25%	-4%	-8%	-10%	-8%	-7%	-5%	-6%	-5%	-2%

Fig. 22. Summary of the optimization effect (W-aware: Workload-Aware; F-based: Frequency-based).

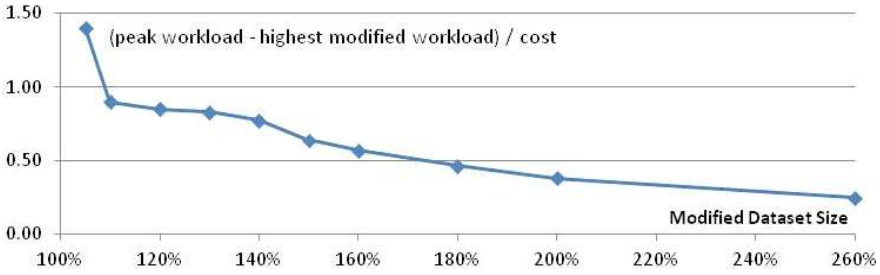

Fig. 23. Ratio between the percentage workload reduction and materialization cost.

workload-aware and the frequency-based selection have the same effect on workload reduction (12%) and the total number of optimized requests (4% difference). This can be explained by the existence of candidate views that fit in the given cost limit while being preferred by both algorithms. A similar situation occurs when the cost limit approaches 140% of the original size. At that point, the majority of views is selected and both selections start to produce similar output. However, the optimization effectiveness (in terms of the ratio between the produced effect and the materialization costs) decreases with size. Therefore, the situation where a majority of views are allowed to be materialized is not expected to happen in a production environment where the cost limit should be set to a more optimal level (i.e. in the range from 20% to 50%).

6.6. *Optimized workload*

Allowing the database to double in size (with cost limit set to 100%) results in a further decrease of the peak workload (see Fig. 24). While the distinction between the effect of both selection methods is less visible, the workload-based approach still offers higher peak reduction while affecting only 5% less requests than the frequency-based approach.

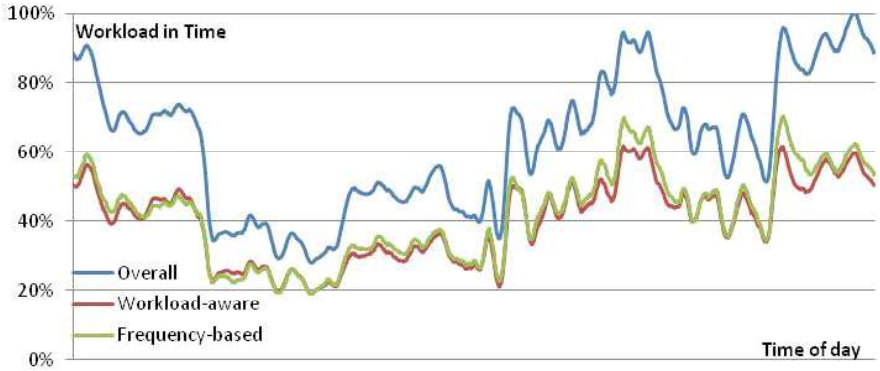

Fig. 24. The original system workload and modified workload with cost limit = 100%.

Cost	5%	10%	20%	100%
Workload aware	0.538	0.373	0.470	0.575
Frequency based	0.095	0.355	0.271	0.134

Fig. 25. Correlation between the original workload.

Figure 25 shows the correlation between the workload reduction and the shape of the overall workload. The workload reduction is the difference between the overall workload and the workload resulting from the optimization.

While not required for the peak workload reduction, the higher correlation measured for the results of workload aware selection suggests that the frequency of candidate views selected by the workload-aware approach matches closely the overall system workload.

6.7. *Discussion and qualitative evaluation*

The experimental evaluation has shown a high optimization effect resulting from the use of materialized views. The experiment involving GREENet data has confirmed that materialization of a large proportion of candidate views allows doubling the original query throughput.

As the decrease in the index access and the subsequent reduction in the I/O operations are expected to have major contribution in the optimization effect, the size of the dataset and size of the main memory available in the test machine has an effect on the evaluation result. Executing tests with a small dataset and large amount of available main memory would allow caching most of the data (either explicitly by the database or implicitly by the in-memory mapping of files provided by the operating system) thus neglecting the effect of the optimization in terms of I/O access. In the opposite situation in which the dataset size is increased (in relation to the cache size), the optimization benefit is expected to be no worse than the effect measured in the experiment.

Before materialized views can be used for optimization, the candidate views have to be extracted and selected views need to be materialized. The time required for views preprocessing is typically not included in the evaluation, as it is a one-time operation and the operation related to views maintenance are considered to be background operations not interfering with normal operation of the database.

Directing the focus of the candidate views selection method to view that reduce the peak system workload has proved to be effective for the investigated real-world dataset. By increasing the dataset size by 40%, the highest modified workload has been reduced by 30% compared to the original system workload. At the same time, the observed peak reduction is over 20% higher than the reduction observed for the more traditional frequency-based selection of views, which confirms the initial expectation.

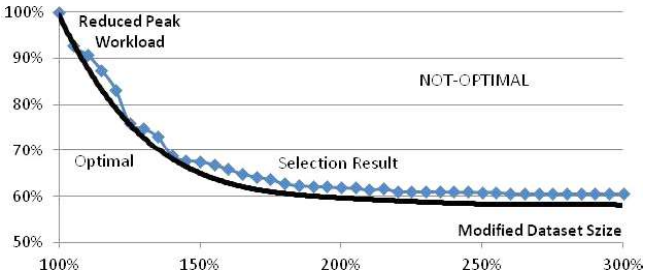

Fig. 26. An example of a possible optimal workload reduction.

The use of a heuristic method and the fact that the materialization cost limit is configurable means that the method cannot guarantee that the selected set of candidates is optimal. Figure 26 shows an example of the optimal workload reduction and the selection results obtained with use of heuristics.

If all possible permutations of candidate views were investigated, any set of candidates placed above the optimal curve should be rejected in favor of candidates located closer to the optimal curve. While the heuristics approach gives only an approximation of the optimal selection, the selection algorithm guarantees that the selection results are at least 50% as efficient in the worstcase scenario (see Sec. 4). Naive search for an optimal solution is infeasible in the publicly accessible dataset, as the number of possible candidate views is prohibitive.

When compared to the frequency-based selection, the total number of affected query requests is lower for the views selected with use of workload-aware approach (by 4% to 8% in most of the evaluated cost range). Maximizing the total number of the optimized queries and the consequent minimization of the total execution time are the usual factors analyzed in existing approaches to views materialization (e.g. Castillo and Leser in Ref. 7). While these two factors are beneficial in scalable environments where the operational cost can be proportional to the total execution time (e.g. Data as a Service), the reduction of the peak workload can be more beneficial in other applications. By reducing the amount of computations required during the highest workload, individual hosts and server clusters can support a higher number of concurrent users or offer a better quality of service during the times of peak usage. Note that the frequency-based selection also affects the peak workload; however, as it is not being targeted explicitly, the resulting reduction is not as high.

Analysis of the optimization results on the final users' needs to include the fact that, while views materialization gives overall reduction in the time needed to execute a group of queries, individual users could suffer higher response times for queries accessing data structures for which no views were materialized. The evaluation results show, the execution time grows by nearly 20% (depending on the dataset increase resulting from the materialization of views). However, with the average execution time of a query measured in millisecond, the additional delay caused by the

database is insignificant for an individual query (especially if effects of additional network delays were to be included).

While the execution time of different groups of queries was measured in the first phase of the experiment, part of the evaluation results used to compare the workload changes in the second phase of the experiment is using the number of queries rather than the actual execution time. This method is appropriate for showing benefits of optimization techniques that include data prefetching in Ref. 32. As the actual execution time is not measured, the results are not affected by the parameters of the machine used in the evaluation.

7. Conclusion

Increasing numbers of organizations publish their data or documents in open linked data to increase their reuse. The system hosting these large structured data repositories with multiple users accessing at the same time could have QoS issues when the peak workload occurs. The proposed view materialization selection method is to address the issue by analyzing the relationship between workload patterns and queries to provide an objective measurement for the selection of view candidates in order to maximize optimization benefits. The implemented optimization system operates as a proxy system and does not require tight integration with SPARQL engines (other than the basic API access). This feature allows it to be used on top of existing databases, without the need for increasing their complexity.

The main contribution of the proposed workload-aware materialization selection method is able to effectively and efficiently address two key properties, velocity and volume, in growing big open linked data and satisfy QoS.

Acknowledgment

This work was supported in part by grants from EU Framework Programme 7(FP7) Project DEHAMS grant No 224609 and GREENet grant No 269122.

References

1. EU Open Data Portal, Available at https://open-data.europa.eu/en/linked-data (2016).
2. Y. Jiang, A survey of task allocation and load balancing in distributed systems, *IEEE Trans. Parallel Distrib. Syst.* **27** (2015) 585–599.
3. T Neumann and G. Weikum, The RDF-3X engine for scalable management of RDF data, *VLDB J.* **19** (2010) 91–113.
4. W3C: RDF/OWL representation of WordNet, Available at http://www.w3.org/TR/wordnet-rdf/ (2006).
5. S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak and Z. G. Ives, Dbpedia: A nucleus for a web of open data, *ISWC/ASWC* (2007), pp. 722–735.
6. F. M. Suchanek, G. Kasneci and G. Weikum, Yago: A core of semantic knowledge, in *Proc. 16th Int. Conf. World Wide Web* (2007), pp. 697–706.

7. R. Castillo and U. Leser, Selecting materialized views for RDF data, in *Proc. 10th Int. Conf. Current trends in Web Engineering (ICWE'10)* (Springer-Verlag, Berlin, Heidelberg, 2010), pp. 126–137.
8. V. Dritsou, P. Constantopoulos, A. Deligiannakis and Y. Kotidis, Optimizing query shortcuts in RDF databases, in *Proc. 8th Extended Semantic Web Conference on The Semanic Web (ESWC'11)* (2001), pp. 77–92.
9. A. Balmin, F. Özcan, K. S. Beyer, R. J. Cochrane and H. Pirahesh, A framework for using materialized XPath views in XML query processing, *The Thirtieth Int. Conf. Very Large Data Bases*, Vol. 30 (2014), pp. 60–71.
10. X. Xu and M. Ozsoyoglu, Rewriting XPath queries using materialized views, in *Proc. Very Large Data Base*, Trondheim, Norway (2005).
11. A. Arion, V. Benzaken, I. Manolescu and Y. Papakonstantinou, Structured materialized views for XML queries, *The 33rd Int. Conf. Very Large Data Bases* (2007), pp. 87–98.
12. B. Cautis, A. Deutsch and N. Onose, XPath rewriting using multiple views: Achieving completeness and efficiency, *WebDB* (2008), https://dblp.org/db/conf/webdb/webdb2008.html#CautisDO08.
13. N. Tang, J. X. Yu, M. T. Ozsu, B. Choi and K. F. Wong, Multiple materialized view selection for XPath query rewriting, *ICDE '08* (IEEE CS, 2008), pp. 873–882.
14. K. Karanasos, *View-Based Techniques for the Efficient Management of Web Data* (Paris-SUD University, 2012).
15. D. Chen and C. Y. Chan, Viewjoin: Efficient view-based evaluation of tree pattern queries, in *Proc. ICDE* (2010), pp. 816–827.
16. T. Neumann and G. Weikum, RDF-3X: A RISC-style engine for RDF, in *Proc. Very Large Data Bases*, Vol. 1 (2008), pp. 647–659.
17. M. Schmidt, M. Meier and G. Lausen, Foundations of SPARQL query optimization, in *Proc. 13th Int. Conf. Database Theory (ICDT '10)* (ACM, 2010), pp. 4–33.
18. M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer and D. Reynolds, SPARQL basic graph pattern optimization using selectivity estimation, in *Proc. 17th Int. Conf. World Wide Web (WWW '08)* (ACM, 2008), pp. 595–604.
19. W. C. Le, A. Kementsietsidis, S. Duan and F. F. Li, Scalable multi-query optimization for SPARQL, in *Proc. ICDE '12* (2012), pp. 666–677.
20. G. Miklau and D. Suciu, Containment and equivalence for a fragment of XPath (ACM, 2004), p. 51.
21. P. Roy, S. Seshadri, S. Sudarshan and S. Bhobe, Efficient and extensible algorithms for multi query optimization, *SIGMOD '00* (ACM, 2000), pp. 249–260.
22. J. W. Raymond and P. Willett, Maximum common subgraph isomor-phism algorithms for the matching of chemical structures, *J. Computer-Aided Mol. Des.* **16** (2002) 521–533.
23. S. Chaudhuri and G. Weikum, Foundations of automated database tuning, *SIGMOD '05* (2005), pp. 964–965.
24. C. Liu, H. Wang, Y. Yu and L. H. Xu, Towards efficient SPARQL query processing on RDF Data, *Tsinghua Sci. Technol.* **15** (2010) 613–622.
25. S. Chaudhuri and V. Narasayya, AutoAdmin "what-if" index analysis utility, *SIGMOD '98* (1998), pp. 367–378.
26. R. Kaushik, P. Shenoy, P. Bohannon and E. Gudes, Exploiting local similarity for indexing paths in graph-structured data, *ICDE'12* (2012), pp. 129–140.
27. V. Harinarayan, A. Rajaraman and J. D. Ullman, Implementing data cubes efficiently, *SIGMOD'96* (1996), pp. 205–216.
28. P. Kilpelainen, On the approximation ratio of Greedy Knapsack (2014), University of East Finland [online], Available at https://www.cs.uku.fi/~kilpelai/ASA/greedyKnapack_0.5_approx.pdf.

29. T. Zlamaniec, K.-M. Chao, N. Godwin, N. Shah and R. Farmer, A framework for workload-aware views materialisation of semantic databases, *ICEBE 2015* (IEEE CS, 2015), pp. 15–22.
30. OpenLink Software: DBPedia Log Files, (2011), Available at ftp://download.openlinksw.com/support/dbpedia/.
31. M. Morsey, J. Lehmann, S. Auer and A. C. S. Ngomo, DBpedia SPARQL benchmark: Performance assessment with real queries on real data, *ISWC'11*, Vol. 1 (2011), pp. 454–469.
32. J. Lorey and F. Naumann, Detecting SPARQL query templates for data prefetching, in *Proc. 10th Extended Semantic Web Conference (ESWC)* (LNCS, 2013), pp. 124–139.